A Novel Approach to the Detection of Cheating in Multiplayer Online Games

Peter Laurens*, Richard F. Paige*, Phillip J. Brooke‡, and Howard Chivers§
*Department of Computer Science, University of York, UK
‡School of Computing, University of Teesside, Middlesbrough, UK
§Cranfield University, Shrivenham, UK
Email: [laurens,paige]@cs.york.ac.uk, P.J.Brooke@tees.ac.uk, hrchivers@iee.org

***Abstract*— Modern online multiplayer games are complex heterogeneous distributed systems comprised of servers and untrusted clients, which are often engineered under considerable commercial pressures. Under these conditions, security breaches allowing clients to employ illegal behaviours have become common; current commercial approaches have limited capabilities for reacting rapidly to such threats. This paper presents an approach to the detection of a cheating player, and describes a proof-of-concept system designed to detect cheating play (specifically wall-hacking) through the analysis of player behaviour. This approach differs from current methods in that it does not rely on knowledge about specific vulnerabilities and their method of exploitation in order to protect the system, but instead monitors player behaviour for indications of cheating play. Statistical evidence is presented which shows that the proof-of-concept correctly distinguishes between most cheating and non-cheating players.**

I. Introduction

In conventional sporting games, rule breaking may take the form of doping, i.e., through use of prohibited chemicals such as steroids. In board games, rule breaking often takes the form of collusion amongst players. And in complex online computer games, involving multiple servers and untrusted, heterogeneous clients, cheating manifests itself in curious virtual versions of these real-world counterparts, e.g., 'virtual doping' in the form of *aimbots* and *speedhacks* [1], and 'virtual collusion' in the form of *ghosting* [6]. But the possible bounds of cheating extend much further in online games than in the real-world. In an online game environment, the laws of the game world are fundamentally more arbitrary and mutable, allowing cheating users to afford themselves extra-sensory abilities and advantages such as seeing through walls (*wallhacks*), or learning about the exact state of other players in the world (*information exposure*). Any game or sport we conceive of and host in the real world is bound by the immutable laws of our universe. We cannot be so confident of the non-existence of exploitable loopholes, or of the stability and reliability of the laws of a complex online game world architected under commercial pressures. Thus, through inevitable security loopholes and oversights, cheating has become rife in a significant number of online games. Clearly, if we are to improve confidence in, and the reputation and enjoyability of online-gaming, as well as promote it as a legitimate and mature pastime (or even profession) we must work to protect the integrity of a game and deter or prevent players from cheating.

Online game developers face an up-hill battle in combating the cheat developers. Cheat developers are able to develop and perfect an exploit in privacy and then, when ready, release it to the public where it may be used immediately. From this moment on, players are free to engage in cheating via this provided exploit, before its existence may even be known to the game developer. The result of this is that the game developer always acts defensively, reactively countering the exploits made public by the cheat developers. The reactive nature of this cycle means that there is a considerable amount of time available to cheaters where there is no method of detection or defence against their behaviour. It takes time to discover, understand, produce, and test a patch for a game. Meanwhile, the install-base remains vulnerable to attack. No repository of game-exploits is currently available in order to formally determine the average time between exploit and subsequent patch, but [23] and [24] provide access to supposedly 'VAC-proof' (unpatched) cheats, and frequently show active cheats unpatched for several months. It is likely that exploits commonly remain undetected for even longer periods of time than this.

It is the fundamentally defensive nature of the game-developer's role which gives cheat developers their biggest advantage; while the game-developer works to understand the current exploits in the wild, there are already new mechanisms for cheating under development. This is a battle which is very difficult for the game developer to win, and it appears that the issue requires the introduction of a disruptive technology or technique in order break this cycle.

The work described in this paper investigates the technical and life-cycle deficiencies which may permit or exacerbate unwanted cheating activity. A proof-of-concept system is presented that attempts to circumvent these deficiencies via detection thorough behavioural analysis, which also abstracts the provision of cheat detection, allowing it to be generically applicable across different online games and game-engines without significant customisation. An implementation for Valve Software's Source (*Half-Life2*) engine running *CounterStrike: Source* [20] is outlined, allowing the approach to be quantitatively evaluated in a real-world production game environment.

12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)
0-7695-2895-3/07 $25.00 © 2007 IEEE

II. Current Methods of Cheating and Counter-measures

This section will look briefly at the current state of cheating in online games in terms of: (a) common techniques used to implement game-exploits; (b) illegal in-game advantages, or *cheats* facilitated by those techniques; (c) commercial methods and tools currently employed to provide protection against cheating play; and (d) an overview of research contributions.

A. Common Cheat Techniques

The practical technique used to implement a cheat may take a number of different forms which differ in complexity and effectiveness. The cheats (or 'hacks') detailed here involve the modification of the client computer, either in the form of changing the client game code, or the client game environment (e.g., drivers). As such, they may be categorised under Yan's [1] taxonomy of cheats, and also using the analysis from [11].

1) Hard-coded Hacks: Hard-coded hacks are considered rudimentary, and are implemented by replacing code in the installed client files, e.g., changing a DLL file to an altered one. Such changes may improve aiming on the client-side in the form of an aimbot, or change the engine code to render walls semi-transparently. They are very tightly coupled to the game-code, making them easy to detect; for example, asking the client to create a hash of the local game-code will clearly show whether the client code has been altered.

2) External Hacks (Fig. 1): An external hack exists as a separate process to the game client, and does not alter the client code directly. Instead it attempts to affect the play of the game through legitimate channels. An early example of such a hack involved changing the player model skins in the game so that all enemies would appear flat-shaded red, the external hack would then monitor the output of the graphics driver, constantly scanning for red-pixels. When red pixels (which denote an enemy) were detected in the output, the hack would attempt to move the player's mouse via the standard Windows API so that the player in the game would aim automatically at the enemy [12]. External hacks are considered basic; the example described had a tendency for jerky movement, and to erroneously lock-on to innocuous parts of the environment.

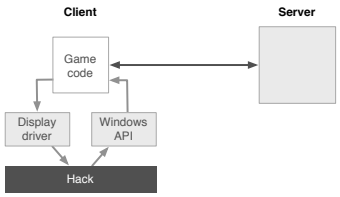

Fig. 1. External hack

3) Environment Hacks (Fig. 2): Environment-based hacks involve altering the computing environment in which the game client runs in order to cheat. It differs from hard-coded hacks in that the game client code is left unaltered; it differs from external hacks as no unique and independent process is involved. A typical environment-based hack may involve the altering or replacement of the client system's video driver in a way that makes rendered objects semi-transparent.

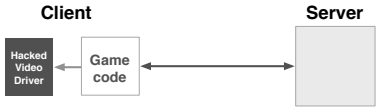

Fig. 2. Environment hack

4) Hook Hacks: Hooks involve running the game from inside a launch harness process, rather than from the game's main executable. This launch harness, or 'hook', can read, control, and inject data into game memory locations. The hook is programmed to be aware of important memory locations, such as player positions, and so can access this data at run-time to the advantage of the cheating player.

5) Packet Tampering (Fig. 3): Packet tampering involves inspecting packets as they leave the client, and altering their contents where it is determined more optimal play could have occurred. The client game-code and environment remains completely unaltered. An implementation normally consists of an entirely separate machine acting as a proxy server to the game-client system. The proxy server will run a sophisticated program which inspects incoming and outgoing packets and learns about the state of the game-world from them. In this way, the proxy builds up basic information such as the position of players in the world, and the orientation of the cheater. If it detects an outgoing command-packet where the player has attempted to shoot an enemy player but missed, the proxy program may make alterations to the packet that indicates a different orientation of the shooting player, one that would have lead to a correct hit. The game server has no reason to disbelieve this packet, making packet-tampering techniques very difficult to detect, but also highly complex to implement.

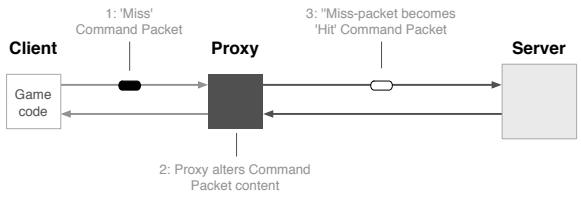

Fig. 3. Packet-tampering hack

B. Common Cheat Implementations

The mechanisms outlined above allow a host of illegal advantages to be bestowed upon a cheating player and have been used to implement a great spectrum of such 'cheats'. This section takes a brief look at the next level of the game exploitation process, that of common cheat implementations:

1) Wall-hacking: Wall-hacking is the name given to cheats which provide the player with the ability to see through walls in some capacity. The benefit to the cheating player is that he can assess the location and movement of opponents to great

advantage, such as preparing an ambush or pre-aiming at an approaching opponent.

2) Aim-bots: Aim-bots provide the player with a computer-augmented ability to aim at other players. All first-person shooter (FPS) games rely to some extent on the reaction times and accuracy of the player, an aim-bot removes these 'handicaps' from the player by allowing users to aim immediately and automatically at opponents.

3) ESP: Extra-Sensory Perception is the name of a genre of information exposure cheats which involve divulging game information to the player which he should not have access to, and which awards him an unfair advantage over opponents. For example, 'Map' ESP in the online game *Counter-Strike* simply involves showing enemy locations on the in-game map by way of a hook into the game memory which extracts the client's 'behind-the-scenes' information about the whereabouts of other players.

4) Bang-hacking: Bang-hacks provide the player with immunity to the effects of blinding grenades in FPS games.

5) Cross-hair Cheats: In some FPS games certain weapons which are intended to be used at long-range do not draw a cross-hair to the screen. This works to prevent players from using very powerful weapons accurately at short range, and helps balance the gameplay. Cross-hair cheats reinstate the cross-hair, either through a hook to the game-code, a hacked version of the video-driver, or even an external process which draws a cross-hair dumbly in the dead-centre of the screen.

6) Content-based Cheats: Content cheats involve modifying the game content (e.g., textures) rather than the actual engine code. For example, enemy textures may be altered so that they stand out against the world, or enemy footsteps made artificially louder.

C. Some Relevant Cheat-Prevention Techniques and Tools

Several cheat-prevention/detection tools have been developed, each of which employ an array of techniques in order to try to prevent illegal play. The four major commercial tools in use at the time of writing are: Valve Software's [20] '*VAC*' (Valve Anti-Cheat) [18], Even Balance Inc.'s [21] '*PunkBuster*' [16], United Admin's [22] '*Cheating-Death*' (now discontinued) and '*HLGuard*' [19]. These developers do not make public the full range of anti-cheat techniques that their tools utilise, but the following list summarises what is known about the mechanisms they employ, taken from the above sources:

1) Memory-scanning: Involves checking the contents of memory used by the game in real-time for the presence of known cheat-hooks, or evidence of tampering with the memory from outside the game-code.

2) Authorisation-servers: Provides a single, trusted account server used by game-servers to indicate which users are trustworthy, and therefore allowed to connect. The users are identified by WON number or CD-key on Punkbuster authentication servers, and by SteamID on VAC authentication servers. This allows authorised game servers to ensure that players connecting have not been blacklisted for cheating in the past on any other authorised server.

3) Content-hashing: In order to detect alterations to game files or game content, the Punkbuster or VAC server may ask the client for a hash of a particular set of game files, and compare this hash against what is expected.

4) Screenshotting: Administrators on Punkbuster enabled servers may request a screen-shot of any player's screen at any time, the results of which may allow the administrator to visually confirm suspicions of wall-hacks or other attacks.

5) API/Driver Scans: The anti-cheat package *Cheating-Death* is known to scan for activity in the graphics driver directly before launching a game in its own launch harness (or hook), thereby monitoring for driver and API-based exploits.

6) Delayed bans: VAC implements a controversial policy of delaying bans for some weeks after a cheating player has been found and confirmed to be cheating. The reasoning for this is that Valve believes more cheaters can be caught if time is given for a cheat to spread.

7) Anti-wallhack: VAC has an inbuilt anti-driver-wall-hack mechanism which checks whether opponents should be occluded by walls or objects in the gameworld, and will not draw them if they are. This means that player models never even get sent to the graphics driver, and so tampering with the driver will not reveal the locations of opponent players.

8) Anti-crosshair: Cheating-death checks the window stack to ensure that no program is running above the game in the stack, so that cross-hairs for sniper weapons cannot be provided in this manner by an external application.

D. Existing Research

A large proportion of existing academic work on cheating and security for online gaming has focused on providing a secure environment that makes peer-to-peer (P2P) based gaming more feasible. There is interest in a pure P2P-based game topology as it promises to reduce the cost overheads and reliability issues of running complex and expensive central servers. Despite these potential advantages, there is a resistance to embracing P2P network architectures in multiplayer online games, primarily because of the issue of *information exposure*.

Chambers et al. [4] describe the issue of cheating in games without a trusted non-biased central server and propose a 'bit-commitment' scheme by hashing sensitive data and a secret together. In this way clients can determine whether or not the opponent was cheating after a game by verifying the final known positions of the objects with the initial hash. This method, however, means the cheating verdict can only be ascertained at the end of a game, and it is not clear that this solution is a scaleable one. Similar work on preventing cheating by engineering specific communication protocols between a client and a central server has been undertaken by both Chen et al. [9] and also Gauthier-Dickey et al. [10], who have investigated the effects of latency and other communication artefacts on game-play in traditional client-server systems and peer-to-peer systems respectively.

Kabus et al. [5] present a higher level approach to securing a P2P based network architecture in larger scale situations such as an MMOG (massively-multiplayer online game) rather than smaller-scale RTS (real-time strategy) games. They propose a more complete solution that ensures that no peer maintains game-world information that is applicable to the current location/actions of the player at that peer. This means that information exposure is still possible, but less applicable to the local player. Rather than fix the fundamental issue of information exposure, Kabus' work attempts to minimise the impact of the technique. Nevertheless, this does not preclude the use of external tools for collusion amongst players, and can break the benefits of locality of interest of the P2P architecture.

There appears to be limited previous academic work addressing the issues of gaming security and cheating specifically. Yan et al. [1][6] have provided a great deal of the foundation work, identifying and classifying cheating behaviour. Also, Brooke, Paige, et al. [7] have developed a conceptual overview of fairness and legality in virtual societies, which Foo and Koivisto [8] have also looked at from a practical angle, investigating 'griefing' (bad) behaviour.

III. A Behavioural-Monitoring Approach to Automated Cheat Detection

Existing commercial tools as outlined in the previous section are severely limited in their ability to detect and deter cheating play. The fact that cheaters can move easily from one exploit to another as older cheats are detected has, for example, forced Valve to adopt a *delayed ban* system, where detected cheaters are not banned immediately, in the hope of catching more illegal players. Current techniques are also poorly automated, requiring manual intervention (for example: administrators needing to request and manually check screenshots for evidence of cheating), are resident on the client machine which is inherently untrustworthy, or are highly game-specific and represent a non-portable investment of development resources (for example *HLGuard*).

Most of these flaws stem from or are exacerbated by a product lifecycle which gives every available advantage to the cheat-developers. This lifecycle can be described as follows: it is inevitable that a game will be released to market with flaws, which will be exploited by cheat-developers over time to enable various types of cheating. These exploits will be detected by the vigilant developer, and usually in a matter of days or weeks a client-server patch is issued to prevent this particular method of exploitation from working. In conjunction with this, cheat-detection systems may be updated on the servers so that further attempts to use this particular exploit may result in the banning of an individual from play.

Figure 4 shows this cyclical process, demonstrating the considerable portion of time that a game is vulnerable to cheating players during the product's life. In reality this is an over-simplification; several flaws may be exploited concurrently. The presence of concurrent exploited flaws threaten to extend the proportion of time that the game is vulnerable considerably.

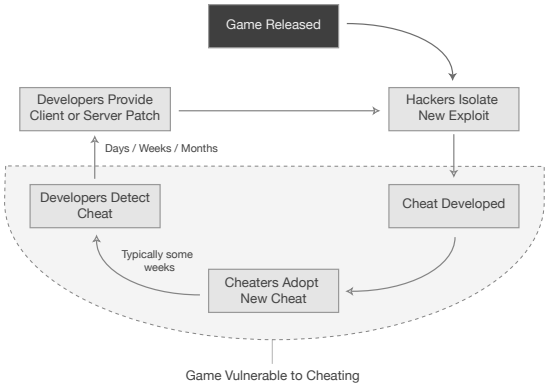

Fig. 4. Traditional exploit-patch lifecycle of a game

As Figure 4 suggests, the current practice for dealing with exploits can be considered to be flawed:

- Most patches are developed to correct flaws that have already been exploited in the wild (so-called *zero-day* exploits), making the Patch-Exploit cycle entirely reactive in its cheat-prevention. This ensures that the game-developers maintain a perpetual disadvantage.
- Patching a game is expensive in terms of the monitoring of the use of cheats, the development time and effort into producing a corrective patch, as well as the distribution costs of getting it to the players.
- In the rare cases where patches *are* proactive (fixing un-exploited flaws), they can identify a flaw to cheat-developers that had previously been undiscovered. The cheat-developers may then take advantage of the fact that it can take some time for all clients and servers to be updated to this latest patch.
- Patches may introduce new flaws of their own.
- Some systems (VAC) delay the banning of cheaters who have been detected by a number of weeks, so that the cheating community is less likely to be alerted when a cheat has become detectable, theoretically allowing more cheaters to be caught. Valve determines this to be necessary to catch more cheaters and prevent them from moving to new undetected cheats quickly, but it allows cheaters to cheat against legitimate players online for a significant amount of time before being banned.

The solution that this paper describes is that of real-time behaviour-based cheat detection. This concept relies on the real-time monitoring and analysis of players' movement and behaviour in the game world, and is based on the central hypothesis that players engaged in cheating exhibit behaviour which is significantly distinguishable from normal play. If this is the case, the cheaters may be identified without regard for their chosen *method* of exploitation. A system based on behavioural analysis to detect cheating play promises significant advantages over current anti-cheat mechanisms:

- It doesn't matter how the cheater is cheating. They may use a hook, a driver exploit, a highly complex packet-

mutating proxy, or a hitherto undiscovered method - it makes no difference to the ability of a behaviour-based system to detect a cheater, as it operates via investigation of the symptoms of cheating behaviour, rather than the exact technical method used.

- Unknown future exploits offer no advantage over known exploits. There is no benefit to the cheater in using an exploit unknown to the developer - the behavioural system should detect all wallhacks/aimbots/etc. equally via their symptoms rather than their particular implementation.
- A cheater may be intimately aware of exactly how the behavioural system works, and what characteristics it is monitoring. It is likely to still have a significant effect on a player's ability to cheat by forcing the player to refrain from *using* his advantage (such as pre-aiming, shooting through walls, tracking enemies through walls, etc.).
- There is less pressure on the developer to issue patches; it is believed that common cheat techniques can be detected automatically and the behaviour-based cheat detection engine may be improved at the developer's pace.
- The behavioural detection system may be run entirely on the server, removing the possibility of tampering with the client code, and pre-empting any man-in-the-middle or proxy attacks. Also, the added resource-burden of the behavioural analysis process is placed fully on the dedicated server rather than the clients.
- A behaviour-based cheat detection system provides a generalised and abstract method of cheat deterrence. Security flaws and their associated exploits in contrast are *uniquely specific* to a particular game or game engine, and require an equally unique solution to fix. A behaviour-based system may provide a level of protection to an entire *class* of online game, with minimal re-engineering required to be applicable to new games and engines as they are developed.

Behaviour-based cheat detection is strongly related to intrusion detection. Such systems rely on two strategies to identify security incidents; the first is signature recognition, which compares data extracts from known exploits with incoming data, and the second is behavioural classification, which uses heuristic analysis to identify anomalous behaviour in networks, host software or applications. Examples include honeypots [15], systems with no operational purpose, allowing any outbound connection to be regarded as a potential anomaly, and host-based anomaly detection systems, where heuristics based on the structure and behaviour of programs are used to detect potential threats [17]. Anomalous behaviour can also be identified at the application layer; for example, parameters used to invoke web applications can be profiled and used to detect unusual web-based requests [13]. These all demonstrate the potential value of behavioural analysis as a means of identifying attacks, and suggest the need to combine a range of heuristics for acceptable false-positive performance.

IV. Approach

A proof-of-concept system was developed as a Source-engine server plugin, and was designed to implement cheat-detection through the behavioural analysis concepts which have been outlined in the previous sections. The goal of the proof-of-concept system was to provide the most simple implementation possible that would demonstrate the effectiveness and feasibility of this concept. In order to achieve this aim, the system concerns itself with only a single type of cheat, *wall-hacking*, in first-person-shooter games.

The proof-of-concept behaviour-based cheat detection system resides on the game-server as a Source-engine plugin, from where it may monitor the actions of players within the game world. Figure 5 shows a high-level overview of the architecture of the cheat detection system as implemented. The analysis engine is componentised and abstracted away from any first-hand interaction with the game-server specifics. Game-server communication takes place via an interaction layer comprised of an input component and an output component. The input component is designed to be engine-specific, it will interface directly with the game-engine pulling player data and performing world-traces upon the state of the game to determine the required information. This is formatted and passed to the analysis engine. The output component is responsible for taking the outputted cheat-score for a player, and passing this back to the game-server so that it may be acted upon.

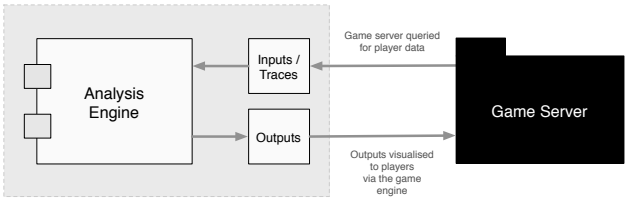

Fig. 5. Architecture for the proof-of-concept system

This architecture allows for a high level of portability - only the interface components (inputs and outputs) need to be re-written for a new game-engine. Also, by abstracting the analysis engine it is possible to update and improve the detection algorithms without having to affect the rest of the system, or the clients.

A key element to the understanding of how exactly the proof-of-concept plugin detects wall-hacking behaviour involves the concept of a *trace*: The plugin must be able to detect precisely what the player is 'looking' at, at any moment in time, and what that object in the world is. A 'trace' is a mechanism provided by the Source engine which allows us to achieve this, it takes the form of a vector which, when 'run', reports information back about what it has hit in the game-world. Traces may be combined with trace *filters*, which filter the objects that are opaque to a trace, for example; allowing it to pass unhindered through walls and other world-geometry, but returning a hit for entities (players) in the world. The wall-hack-detecting proof-of-concept requires two different types

of trace filter, a *world filter* which only returns data about the base-world geometry that it has hit (*world trace*) and an *entity filter* which passes through all world-geometry as if it weren't there, returning only information about any *entities* it hits, regardless of whether the they are behind a wall in relation to the tracing player (*entity trace*).

To be able to monitor for the characteristic behaviour of a cheating player, it was first necessary to identify and understand the observable *differences* between the play of a legitimate player and a cheater, essentially answering the question 'what observable characteristics would allow a system to tell a legitimate player and a cheating player apart?'. No previous work studying the practical behaviour of cheating gamers is known to the authors, and so the first phase in the development of the proof-of-concept system involved a study of captured trace-data of both a wall-hacking player and a legitimate player playing *Counterstrike:Source* on the same maps under the same conditions. The legitimate player used the standard release of *CS:S* (version 1.0.0.5), the wall-hacking player used the same version augmented with version 1.1 of the *FSKWallhack* wall-hack by '*Felikz*' and '*slayer-es-me*'. Play-data in the form of a series of trace-results was captured from the engine as the users played, and was then reviewed to determine the defining characteristics that may help distinguish a cheater from a legitimate player. This produced the following four metrics which, it was determined, demonstrated noticeable differences between the two:

- **Frequency of illegal traces**: An illegal trace is defined as a trace (vector along a player's line of sight) from the player under observation to an enemy entity where the trace passes through opaque world material before reaching the opponent model. The rationale for this selection is that non-cheating players would trace to enemy players behind walls only by chance, whereas it was hypothesised that wall-hackers may tend to track enemies behind walls, thereby giving themselves the advantage that they had aimed and were ready to fire when the opponent is no longer occluded by world material.
- **Distance to world material**: To a wall-hacker, world material (such as walls) does not occlude the player's view, whereas during play, a legitimate player will tend to focus on the farthest point of the map within his field of view. Therefore, it was hypothesised that wall-hacking players would exhibit strange behaviour such as having unusually small traces to world material (appearing to stare blankly at a wall).
- **Distance to illegal traces**: A legitimate player may make chance illegal traces to players behind walls. The randomness of these traces should make it equally likely that such a trace is to a far away player as to a very close player (after target-size difference for the opponent entity is accounted for due to distance). A wall-hacker, however, may be more likely to track closer enemies as he takes advantage of his ability. Due to this, it was hypothesised that the distance to illegal traces will be on average smaller for wall-hacking players than for legitimate players.
- **Consecutivity of illegal traces**: A legitimate accidental trace is unlikely to last more than the briefest of moments as both players move and look around the game-world. A wall-hacker, however, may have a tendency to continuously track an opponents movements behind world material, building up a string of consecutive illegal traces. It was hypothesised therefore that the consecutivity of illegal traces is much higher for wall-hacking players than for legitimate players.

The capacity of these metrics to distinguish between legitimate and illegitimate play was investigated during a metric-testing phase, where data for the four metrics was obtained and compared for a player while cheating and also playing normally. Two separate maps were used in the test to minimise possible idiosyncrasies of world-layout. The results of the metric-test can be seen in tables I-IV, and figures 6 and 7.

Table I shows the power of differentiation of *illegal trace frequency*, giving a 2.37x difference between legitimate and cheating play on the map *de_Chateau*, and 2.63x in the frequency of traces between the two for the map *cs_Office*. Table II shows the results after the data was processed to allow a grace period of 1, 3, and 6 seconds between a player legitimately seeing an opponent, and being punished for illegally tracking them behind a wall. The rationale for this grace period is that *legitimate* players may track opponents as they run behind a wall for cover, thus tracking them behind world-objects illegally, by giving a grace period of a few seconds after legitimately seeing an opponent it ensures that legitimate players cause fewer false positives. This can be seen in the higher differentiation rates.

TABLE I
ILLEGAL TRACE FREQUENCY METRIC

Map played	Mean trace interval	Difference
de_Chateau normal	20.3s	**2.37x**
de_Chateau wallhack	8.55s	
de_Office normal	16.67s	**2.63x**
de_Office wallhack	6.35s	

Figures 6 and 7 show the results of the world-distance metric as a histogram demonstrating the difference between the player and the closest world object being looked at (normally walls). From this data, it appears that there is a slight tendency for cheating players to have a larger proportion of close-distance world-traces (less than 300 world units) than legitimate players.

Table III shows the power of differentiation between a cheating player and a legitimate one, based on the distance-to-illegal-trace metric. This metric shows that the *legitimate* players illegal traces were on average 0.53x the distance of the same player cheating for the map *de_Office*. This level of differentiation was not present in the *de_Chateau* test, but as more data was available for the *de_Office* test, it is believed

0-7695-2895-3/07 $25.00

TABLE II
ILLEGAL TRACE FREQUENCY WITH GRACE PERIOD METRIC

Map played	Grace Period	Mean trace interval	Difference
de_Chateau normal	1 second	37.4s	**3.4x**
de_Chateau wallhack		10.99s	
de_Office normal		17.86s	**2.48x**
de_Office wallhack		7.21s	
de_Chateau normal	3 seconds	41.8s	**3.6x**
de_Chateau wallhack		11.61s	
de_Office normal		20.0s	**2.77x**
de_Office wallhack		7.21s	
de_Chateau normal	6 seconds	41.8s	**3.6x**
de_Chateau wallhack		11.61s	
de_Office normal		20.0s	**2.77x**
de_Office wallhack		7.21s	

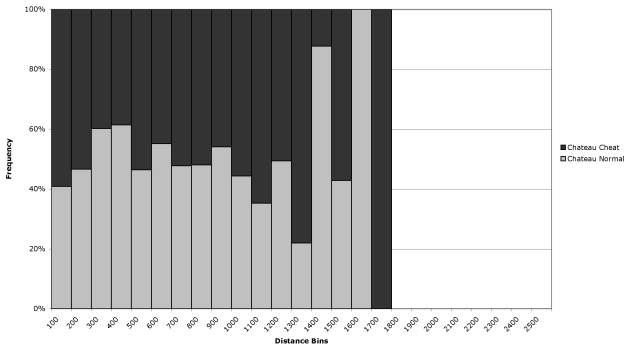

Fig. 6. Comparison histogram of distance to world behaviour, comparing cheating and legitimate players. Map: de_Chateau.

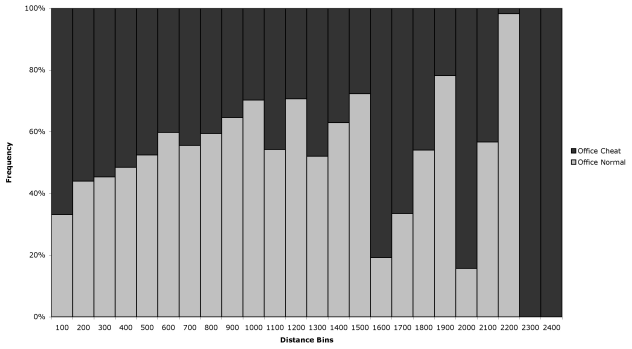

Fig. 7. Comparison histogram of distance to world behaviour, comparing cheating and legitimate players. Map: cs_Office.

that this result is due to too small a data-set for that particular map.

TABLE III
MEAN DISTANCE TO ILLEGAL TRACE METRIC

Map played	Mean Trace Dist.	Difference
de_Chateau normal	844.40 units	**0.99x**
de_Chateau wallhack	834.12	
de_Office normal	1132.90	**0.53x**
de_Office wallhack	604.98	

The consecutivity of traces metric provided the data in table IV, showing a 1.5x differentiation for the map *de_Chateau*, and 1.7x for the map *de_Office*. It was considered, however, that to better distinguish between a legitimate player accidentally illegally tracking an opponent, and a cheating player, consecutive illegal traces would be punished exponentially rather than linearly. This increases the level of punishment considerably for extended strings of consecutive illegal traces, a characteristic that very strongly suggests wall-hacking activity. When this change was made to the metric, the metric's power to differentiate between legitimate and illegitimate players was improved, scoring 3.68x higher for cheating users playing *de_Chateau*, and 4.49x higher for cheating users playing *de_Office*.

TABLE IV
CONSECUTIVITY OF ILLEGAL TRACES

Map played	Mean Number of Consecutive Traces	Difference
de_Chateau normal	1.07	**1.5x**
de_Chateau wallhack	1.6	
de_Office normal	1.08	**1.7x**
de_Office wallhack	1.84	

A non-parametric rank sum test [25] was used (for all of the metrics) to evaluate the null-hypothesis that the data from the cheating player, and that from the legitimate player are from the same distribution. All of the metrics bar one rejected the null-hypothesis at a significance level of 5%. The data that did not meet this level of significance was the distance-to-illegal-trace data for the map *de_Chateau*. The data-sample for this metric was uniquely small, and this is likely to be the cause of the different statistical results for this one set of data in isolation.

An algorithm, drawing on the above data showing the varying ability of the metrics to distinguish cheating play, was developed. The algorithm combines all four metrics giving greater power to those which showed a higher capacity for differentiating. The system records these metrics every 0.1 seconds at runtime, and uses them in the algorithm to provide a probabilistic score that a certain player is cheating. Results from the metrics which are likely to have arisen due to illegal play will cause an increase to a player's cheat-score, persis-

tent abstention from such behaviour will gradually reduce a player's cheat-score. Thresholds can be established where it can be significantly improbable for a player to attain a certain cheat-score through legitimate play, and can be used to mark such a player as a highly probable cheater. The algorithm used to calculate the cheat score for a player is shown in expressions (1) to (6).

$$\begin{aligned} X &= \text{the set of all traces} \\ I &= \text{the set of all illegal traces} \\ X_{player} &= \text{the set of all traces for the player analysed} \\ I_{player} &= \text{the set of all illegal traces for the player analysed} \\ X_{player} \subset X, &\text{ and } I_{player} \subset I \end{aligned} \tag{1}$$

$$a = 60\frac{|I_{player}|}{t} \tag{2}$$

Equation (2) calculates the frequency of illegal traces a player makes, per minute, by taking the total number of illegal traces, dividing that figure by t, the elapsed time of the game in seconds, and then multiplying by 60, to get the figure in minutes.

$$b = a(\frac{\bar{X}}{\bar{X}_{player}}) \tag{3}$$

Equation (3) calculates the world-distance metric, based on the difference in the mean of all the analysed player's traces, and the mean of all combined players.

$$c = a(\frac{\bar{I}}{\bar{I}_{player}}) \tag{4}$$

Equation (4) calculates the illegal trace-distance metric, based on the difference in the mean of the analysed player's illegal traces, and the mean of all combined players' illegal traces.

$$\lambda = \left(\frac{|I_{player}|}{|\{i : x_i \in I_{player} \wedge x_{i-1} \notin I_{player}\}|}\right)^2 \tag{5}$$

Equation (5) calculates the score for the consecutivity of illegal traces for the analysed player. This is done by dividing the total size of the set of illegal traces I, by the number of individual 'runs' of illegal traces (traces whose predecessor was not an element of the illegal trace set), to give the average consecutivity of traces per run. This value is then squared to more harshly punish higher levels of consecutive illegal tracing.

$$\text{Final 'cheat-score'} = b + c + \lambda \tag{6}$$

Equation (6) shows how the final cheat score is determined, by summing the world-distance score, the illegal trace distance score, and the consecutivity of illegal traces score.

V. Preliminary Results

Testing was undertaken with three test subjects, each providing several hours of cheat-score data, generated as described by the algorithm discussed in the previous section. The three subjects were all students aged between 20 and 30 years, and were selected for their experience of online gaming, experience with FPS games, experience with *CounterStrike:Source*, and their basic understanding of the current situation regarding cheats in online games. This was to ensure that the results were close to the play of most normal, competent players, and that they would not be skewed by players unfamiliar with controls or gameplay mechanics etc.

The results are presented in the form of the cheat-scores reported by the algorithm outlined in the previous section, the higher the cheat-point score, the more likely the system 'believes' the player to be cheating. The results are shown in figures 8, 9, and 10. The results show promise in the ability of this technique to distinguish between a legitimate and a cheating player. The system's power of distinction between cheating play and non-cheating play varied widely, at its greatest, a differentiation of 1200x between the scores of the cheating player and non-cheating player (same subject, same map) was observed. At its weakest, it diminished to 1.4x. The mean power of differentiation, across all subjects and all maps was 22x, i.e. on average a player scored 22-times the cheat-score when cheating as compared to when playing normally.

One particular anomalous sample was of note in that it showed a *higher* score for normal play than for cheating play (average of 1.9 points for the normal, and 0.9 points for the cheat). After the samples were taken, the subject was debriefed and asked about this anomaly. He answered that he was intentionally trying to play exactly as he had been before without taking advantage of the cheat, partly out of interest (but also because he found with the wall-hack he was actually performing more poorly in the game than before). This is interesting in that it shows the plugin is essentially capable of detecting when a player is taking advantage of a cheat rather than whether he is using one at all. This is not considered a particular issue with the future applicability of this method of cheat prevention, as it works to prevent a cheater from taking advantage of his illegally obtained abilities - that is, he may be able to see through walls, but he must not use this ability to his benefit. All other samples showed that the plugin could differentiate between the two styles of play. As an aside, it is interesting to consider the difference between preventing cheating *itself*, and preventing *gain* through cheating, which is essentially what is achieved by this technique. It is the opinion of the authors that the two are very often equal in preference.

It is considered important that cheating players may be detected as quickly as possible so that their influence may be removed from the game-world. For the proof-of-concept system, it appears that the time to distinguish a cheating player is between 250 and 350 seconds, as the system needs a certain number of samples to settle. This is still a promising time for an early proof-of-concept, and may be further improved with

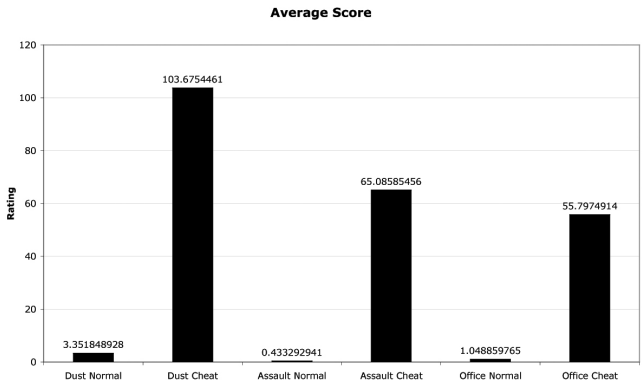

Fig. 8. Score results across six bouts of play for subject 1

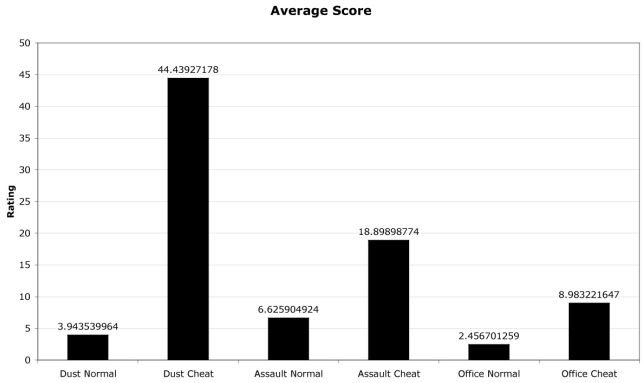

Fig. 9. Score results across six bouts of play for subject 2

a more mature algorithm.

The results show a score-threshold at around 20 points which would distinguish a cheater from a legitimate player. With a threshold in this range, all normal players in the test would be properly identified as non-cheating, and 7 of 9 cheating samples would be properly identified as cheating. False positives, then, appear rare, no legitimate player achieved a higher average score than 6.6. False negatives however appear more likely to cause problems. Some sample data showed a very low score of 0.89 for one game while the subject was cheating (this is the anomaly already discussed). Another two samples show an average score of 8.46, and 8.9, both recorded by cheating players, which would not have been distinguished by a system with a threshold set to 20-points. Overall however, this is not discouraging, with the aforementioned threshold, over 70% of cheating players would be identified.

VI. Future Work

The scope of this work was limited to a focussed and basic implementation, designed to provide a proof-of-concept system that would demonstrate the feasibility of taking this approach to game security. Now that this ground-work has been established, and the results have indicated that behavioural analysis of play may indeed effectively identify the presence

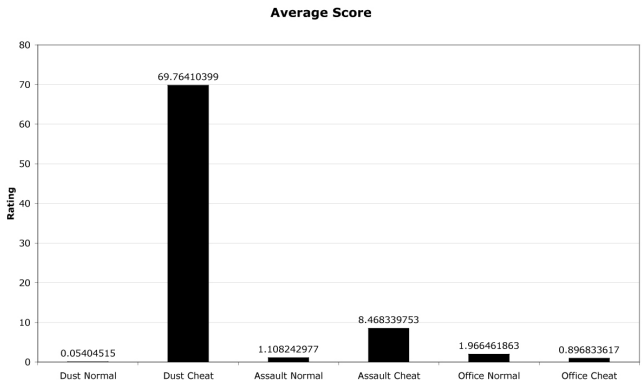

Fig. 10. Score results across six bouts of play for subject 3

of cheating, there is ample opportunity to enhance the proof-of-concept by expanding on the metrics used as a method of identification.

This paper isolated just four metrics to consider: Frequency of illegal traces; consecutivity of illegal traces; distance to world traces; and distance to entity traces. It has been shown that these metrics are indeed able to provide an indication of the likelihood of a player cheating, but no work has been done in assessing the efficiency of these metrics' powers of differentiation, they were simply chosen based the informal monitoring of wall-hacking players, and the fact that they were relatively easy metrics to capture from the engine. This opens two immediate avenues for future work: The assessment and tuning of the metrics used, and the development and assessment of new (more complex) metrics. For example, metrics based on movement and/or speed, metrics involving reaction times of the player after first seeing a new opponent, and more complex *patterns* of play etc. Despite having been shown empirically to distinguish between cheaters and players, the metrics employed have no rigourous statistical foundation, confirmation of their true statistical significance would be desirable in future work.

One may imagine that a more complete behaviour-based cheat-detection system would have many of the characteristics of an *artificial immune system*, such as the ability to classify and distinguish players based on complex interdependent inputs, the ability to recognise cheating play by way of a negative-selection algorithm [2], and an ability to make decisions based on noisy data. An Artificial Immune System approach could also allow for the game-servers to collaborate and communicate their work on classification of players to each other, further enhancing an individual system's ability to distinguish and classify players, work of a similar conceptual nature has been applied to fault detection in ATM machines by Timmis et al. [3].

The work described in this particular paper considers only wall-hacking, but the characteristics of the proof-of-concept are such that it is possible that the system could be engineered to detect other types of cheats. The software could, for example, analyse players for characteristic behavioural elements

which occur when a player uses an aimbot. Such a system would monitor reflexes, the jerkiness of the player's aim, etc. and produce a cheat score based on the likelihood that a player is using an aimbot.

A more advanced and automated way of *capturing* the characteristics of a particular cheat through the monitoring of a known-cheating player would be a great asset. Such a system could produce 'profiles' for different classes of cheat which would be used by the server-plugin. Depending on the complexity and detail of the play-characteristics that could be captured, it may be possible to create profiles which are capable of distinguishing particular patterns and characteristics with great levels of subtlety. It may then be possible to characterise and detect very complex behaviours such as various types of nuisance-play (*griefing*). Some types of cheating, such as information exposure (e.g. enemy positions being shown on a players map) may be much more difficult, or even impossible to detect via a behavioural-monitoring system as proposed by this paper. Such cheats would still have to be combated via existing traditional methods, but it is worth noting that wall-hacks and aimbots are considered to be the most significant and widely deployed cheats in many FPS games. There is also a possibility that an expanded and more flexible system could operate on other game genres, not just the FPS genre that has been the focus of this project. MMORPGs ('Massively-multiplayer online role-playing games') represent a slightly more complex environment with a greater level of social involvement, and would be an interesting genre to investigate. The typical cheats used in an MMORPG may be different to those in an FPS, but it would be interesting to attempt to isolate the behavioural characteristics of cheating users in this different gaming domain.

VII. Conclusion

This paper has explored the struggle to effectively combat security issues in complex multiplayer-games. It is believed that this failure originates from the commercial pressures of game development, which ensures that there will never be sufficient resources available to properly engineer security into what is a complex distributed system, and that cheat-developers will always have the advantage of time and secrecy. We have presented a solution to the problem of detecting tell-tale behavioural *characteristics* of cheating rather than detecting the cheating mechanism itself. In doing so, we have abstracted out the process of providing a certain level of cheat protection for a game, an abstraction which allows a single system to be portable to many different games and many different game-engines with minimal re-engineering and at little cost. This technique has been tested and implemented in a proof-of-concept system, which is designed to detect *wall-hacking*. The system was built to work in conjunction with a modern production game-engine, Valve Software's *Source*, and a modern and popular multiplayer game, *Counterstrike:Source*, demonstrating the validity of this approach though translation of the concept to a production system. The proof-of-concept system, although primitive, showed significant success in its ability to distinguish between legitimate players, and cheating players. This success, the authors believe, provides fertile ground for further work and expansion of the technique, perhaps by way of taking inspiration from work on classification, differentiation, and anomaly detection in the field of artificial immune systems.

References

[1] J. Yan, B. Randell, *A Systematic Classification of Cheating in Online Games, in Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games, October 10-11, 2005, Hawthorne, New York.*, NetGames '05, ACM Press, 2005.

[2] D. Dasgupta, S. Forrest, *Novelty Detection in Time Series Data using Ideas from Immunology*, 5th International Conference on Intelligent Systems, 1995.

[3] R. de Lemos, J. Timmis, S. Forrest, M. Ayara, *Immune-Inspired Adaptable Error Detection for Automated Teller Machines, to appear in IEEE SMC Part C: Applications and Reviews*, IEEE, 2006.

[4] C. Chambers, W. Feng, W. Feng, D. Saha, *Mitigating Information Exposure to Cheaters in Real-time Strategy Games, in Proceedings of the international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '05, ACM Press, 2005.

[5] P. Kabus, W. Terpstra, M. Cilia, A. Buchmann, *Addressing Cheating in MMOGs, in Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, NetGames '05, ACM Press, 2005.

[6] J. Yann, H. Choi, *Security Issues in Online Games*, Emerald Group Publishing, 2002.

[7] P. Brooke, R. Paige, J. Clark, S. Stepney, *Playing the Game: Cheating, Loopholes, and Virtual Identity, in ACM Computers and Society*, ACM Press, 2004.

[8] C. Foo, E. Koivisto, *Defining Grief-Play in MMORPGs: Player and Developer Perceptions, in Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*, ACE '04, ACM Press, 2004.

[9] B. Chen, M. Maheswaran, *A Cheat-Controlled Protocol for Centralised Online Multiplayer Games, in Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, NetGames '04, ACM Press, 2004.

[10] C. GauthierDickey, D. Zappala, V. Lo, J. Marr, *Low-latency and Cheat-Proof Event Ordering for Peer-to-Peer Games, in Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '04, ACM Press, 2004.

[11] Counter-Hack.net, 2006, *How Hacks Work*, viewed June 2006, http://wiki.counter-hack.net/howhackswork

[12] Counter-Hack.net, 2006, *Types of Cheats Associated with Counter-Strike*, viewed June 2006, http://wiki.counter-hack.net/CategoryCSHacks

[13] C. Kruegel and G. Vigna. Anomaly Detection of Web-based Attacks, *Proc. CCS'03*, ACM Press, New York, NY, USA, 2003; 251-261.

[14] M. Leo, T. D'Orazio, P. Spagnolo, *Human Activity Recognition for Automatic Visual Surveillance of Wide Areas, in Proceedings of the ACM 2nd International Workshop on Video Surveillance and Sensor Networks*, ACM Press, 2004

[15] The HoneyNet Project. *Know Your Enemy : Learning about Security Threats*, AWL, 2004.

[16] Even Balance Inc., 2006, *Punkbuster Information*, viewed June 2006, *http://www.evenbalance.com/index.php?page=info.php*

[17] Symantec Inc., *Understanding Heuristics*, Volume XXXIV. September 1997. p. 17. http://www.symantec.com/avcenter/reference/heuristc.pdf

[18] Valve Software Inc., 2006, *Valve Anti-Cheat (VAC)*, viewed June 2006, *http://developer.valvesoftware.com/*

[19] United Admins Limited, 2006, *Cheating Death and HLGuard*, viewed June 2006, http://www.unitedadmins.com/

[20] Valve Software Inc., *http://www.valvesoftware.com*

[21] Even Balance Inc., *http://www.evenbalance.com*

[22] United Admins Limited, *http://www.unitedadmins.com*

[23] Fkn0wned.com, *Downloads*, viewed January 2007, *http://www.fkn0wned.com/*

[24] CSH-Network, *Counter-Strike Source Downloads*, viewed January 2007, *http://www.mirc-scripts.de/*

[25] P. Hoel, *Introduction to Mathematical Statistics, 5th Edition*, John Wiley & Sons, 1984.

0-7695-2895-3/07 $25.00 © 2007 IEEE